**Introduction to Data Visualization with Matplotlib**

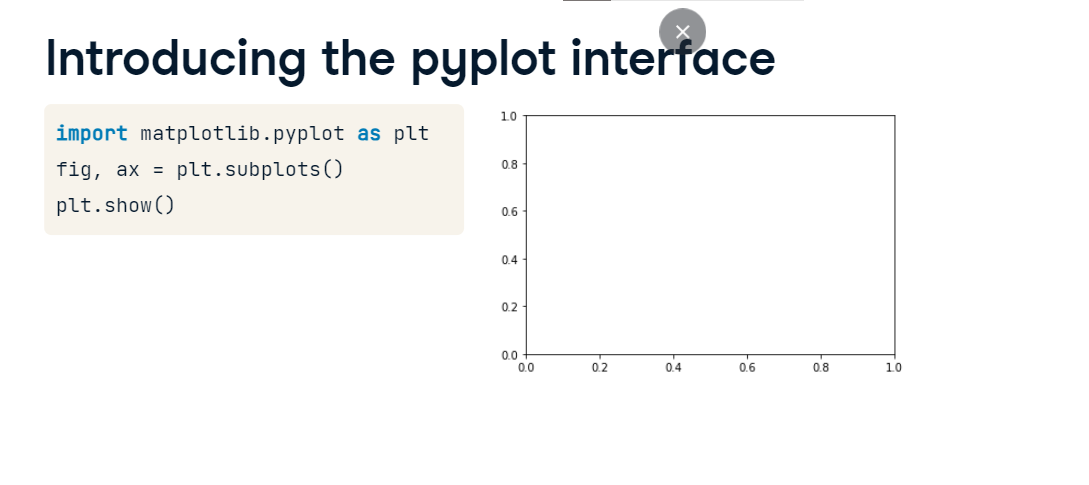
Hello and welcome to this course on data visualization with Matplotlib! A picture is worth a thousand words. Data visualizations let you derive insights from data and let you communicate about the data with others.

**Data visualization**

For example, this visualization shows an animated history of an outbreak of Ebola in West Africa. The amount of information in this complex visualization is simply staggering! This visualization was created using Matplotlib, a Python library that is widely used to visualize data. There are many software libraries that visualize data. One of the main advantages of Matplotlib is that it gives you complete control over the properties of your plot. This allows you to customize and control the precise properties of your visualizations. At the end of this course, you will know not only how to control your visualizations, but also how to create programs that automatically create visualizations based on your data.

**Introducing the pyplot interface**

There are many different ways to use Matplotlib. In this course, we will use the main object-oriented interface. This interface is provided through the pyplot submodule. Here, we import this submodule and name it plt. While using the name plt is not necessary for the program to work, this is a very strongly-followed convention, and we will follow it here as well. The plt-dot-subplots command, when called without any inputs, creates two different objects: a Figure object and an Axes object. The Figure object is a container that holds everything that you see on the page. Meanwhile, the Axes is the part of the page that holds the data. It is the canvas on which we will draw with our data, to visualize it. Here, you can see a Figure with empty Axes. No data has been added yet.



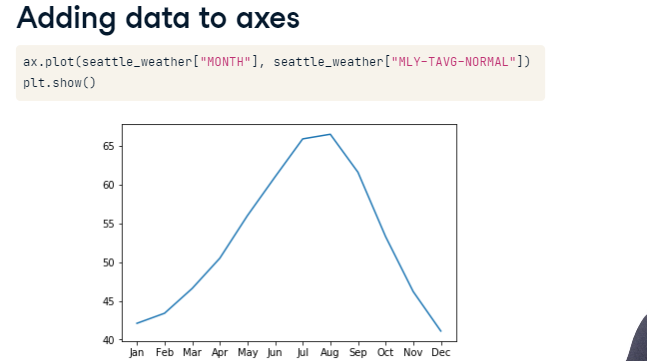
**Adding data to axes**

Let's add some data to our figure. Here is some data. This is a DataFrame that contains information about the weather in the city of Seattle in the different months of the year. The "MONTH" column contains the three-letter names of the months of the year. The "monthly average normal temperature" column contains the temperatures in these months, in Fahrenheit degrees, averaged over a ten-year period.



**Adding data to axes**

To add the data to the Axes, we call a plotting command. The plotting commands are methods of the Axes object. For example, here we call the method called plot with the month column as the first argument and the temperature column as the second argument. Finally, we call the plt-dot-show function to show the effect of the plotting command. This adds a line to the plot. The horizontal dimension of the plot represents the months according to their order and the height of the line at each month represents the average temperature. The trends in the data are now much clearer than they were just by reading off the temperatures from the table.

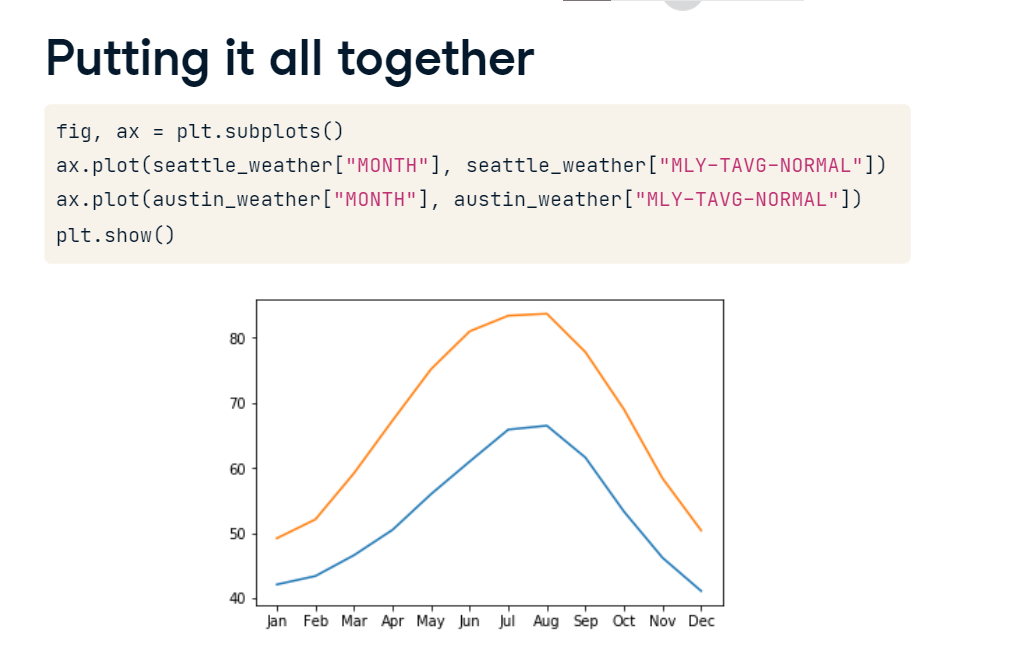


**Adding more data**

If you want, you can add more data to the plot. For example, we also have a table that stores data about the average temperatures in the city of Austin, Texas. We add these data to the axes by calling the plot method again.

**Putting it all together**

Here is what all of the code to create this figure would then look like. First, we create the Figure and the Axes objects. We call the Axes method plot to add first the Seattle temperatures, and then the Austin temperatures to the Axes. Finally, we ask Matplotlib to show us the figure.

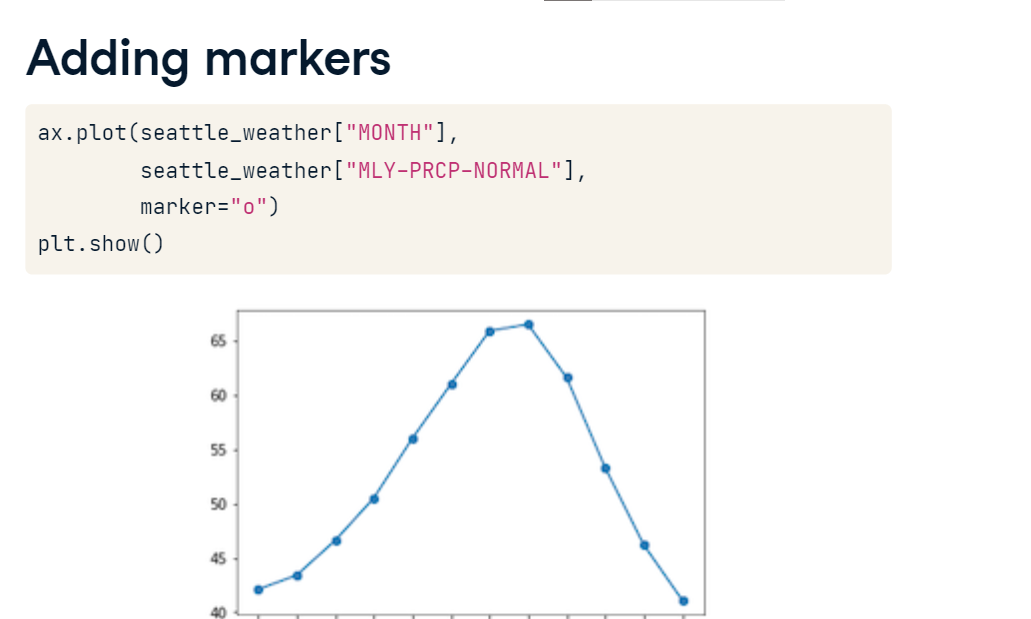


**Customizing your plots**

Now that you know how to add data to a plot, let's start customizing your plots.

**Customizing data appearance**

First let's customize the appearance of the data in the plot. Here is the code that you previously used to plot the data about the weather in Seattle. One of the things that you might want to improve about this plot is that the data appears to be continuous, but it was actually only measured in monthly intervals. A way to indicate this would be to add markers to the plot that show us where the data exists and which parts are just lines that connect between the data points.



**Adding markers**

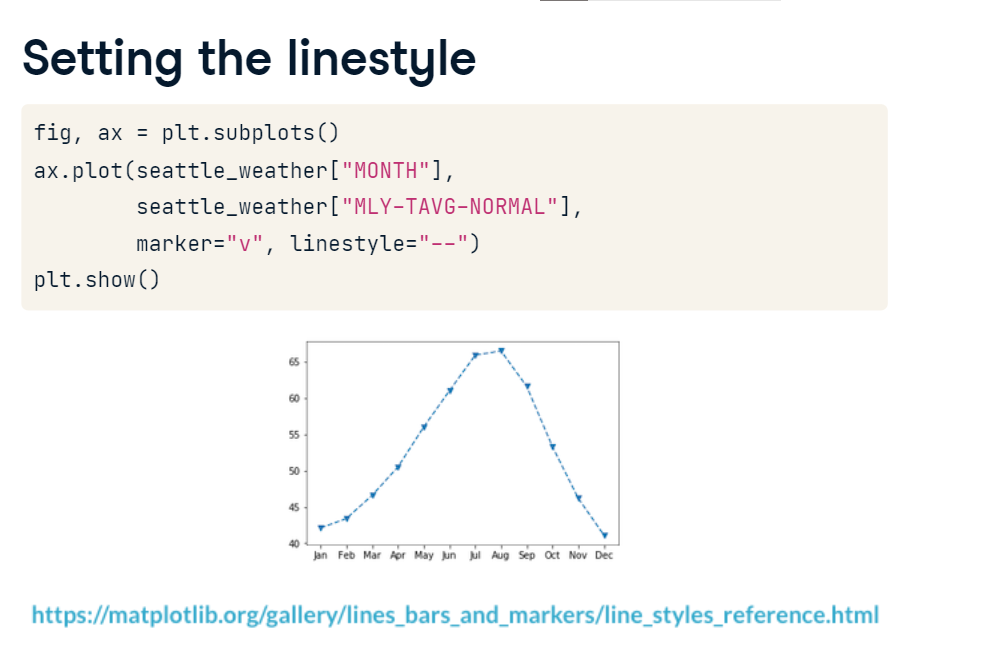
The plot method takes an optional keyword argument, marker, which lets you indicate that you are interested in adding markers to the plot and also what kind of markers you'd like. For example, passing the lower-case letter "o" indicates that you would like to use circles as markers.

**Choosing markers**

If you were to pass a lower case letter "v" instead, you would get markers shaped like triangles pointing downwards. To see all the possible marker styles, you can visit this page in the Matplotlib online documentation. In these versions of the plot, the measured data appears as markers of some shape, and it becomes more apparent that the lines are just connectors between them.

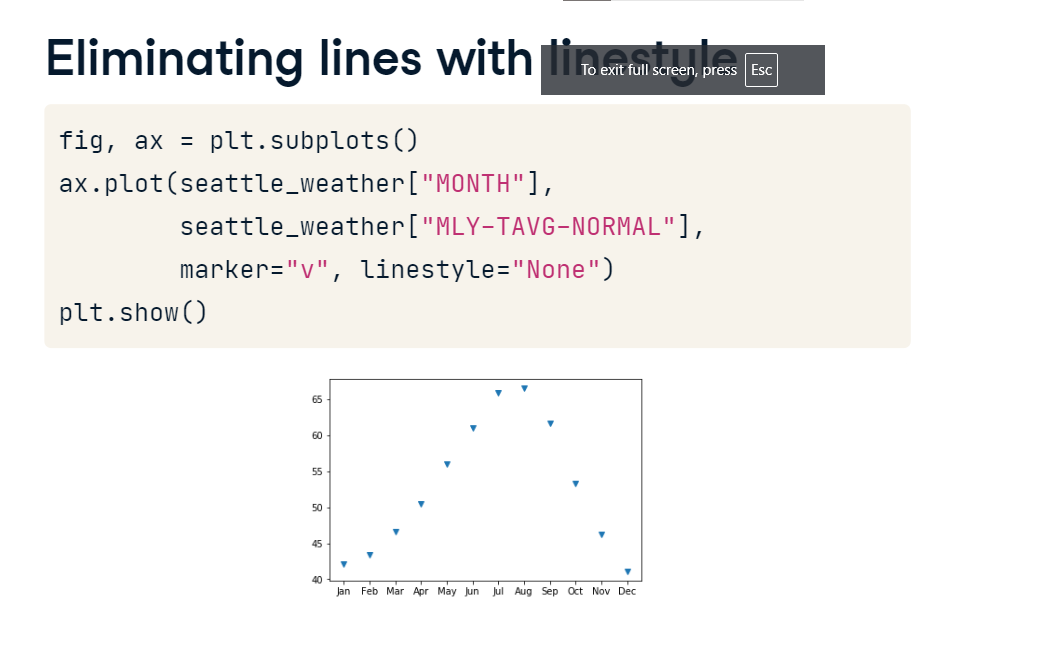
**Setting the linestyle**

But you can go even further to emphasize this by changing the appearance of these connecting lines. This is done by adding the linestyle keyword argument. Here two dashes are used to indicate that the line should be dashed. Like marker shapes, there are a few linestyles you can choose from, listed in this documentation page.



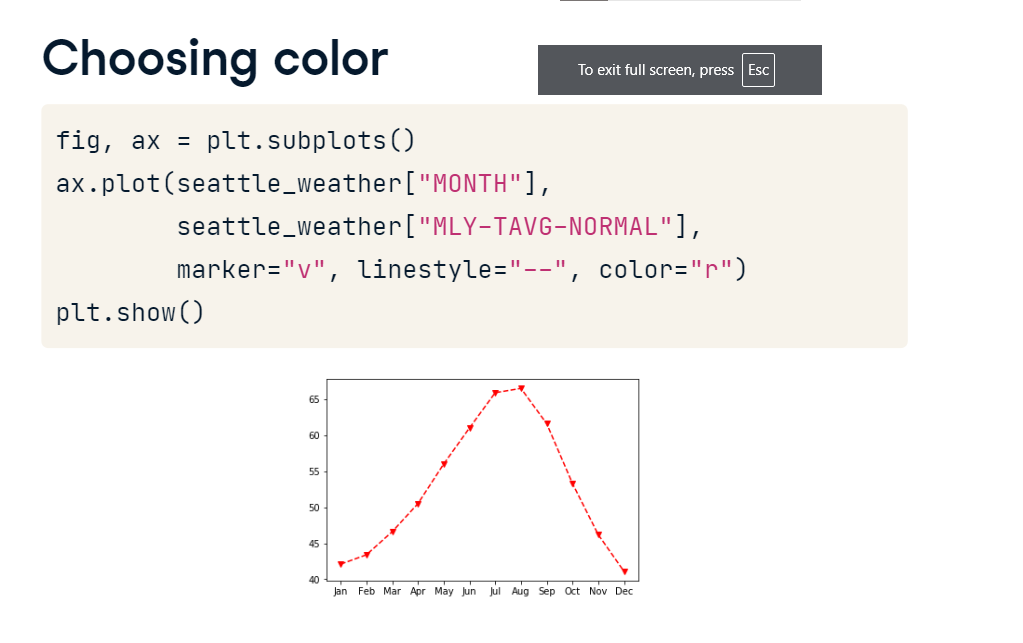
**Eliminating lines with linestyle**

You can even go so far as to eliminate the lines altogether, by passing the string "None" as input to this keyword argument.



**Choosing color**

Finally, you can choose the color that you would like to use for the data. For example, here we've chosen to show this data in red, indicated by the letter "r".

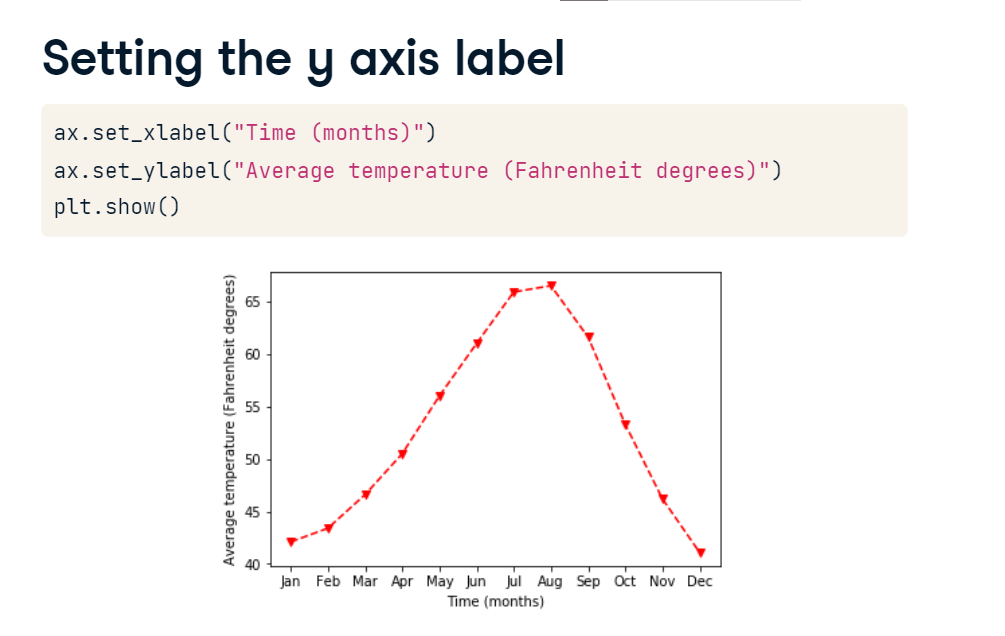


**Customizing the axes labels**

Another important thing to customize are the axis labels. If you want your visualizations to communicate properly you need to always label the axes. This is really important but is something that is often neglected. In addition to the plot method, the Axes object has several methods that start with the word set. These are methods that you can use to change certain properties of the object, before calling show to display it. For example, there is a set-underscore-xlabel method that you can use to set the value of the label of the x-axis. Note that we capitalize axis labels as we would capitalize a sentence, where only the first word is always capitalized and subsequent words are capitalized only if they are proper nouns. If you then call plt-dot-show you will see that the axis now has a label that indicates that the values on the x-axis denote time in months.

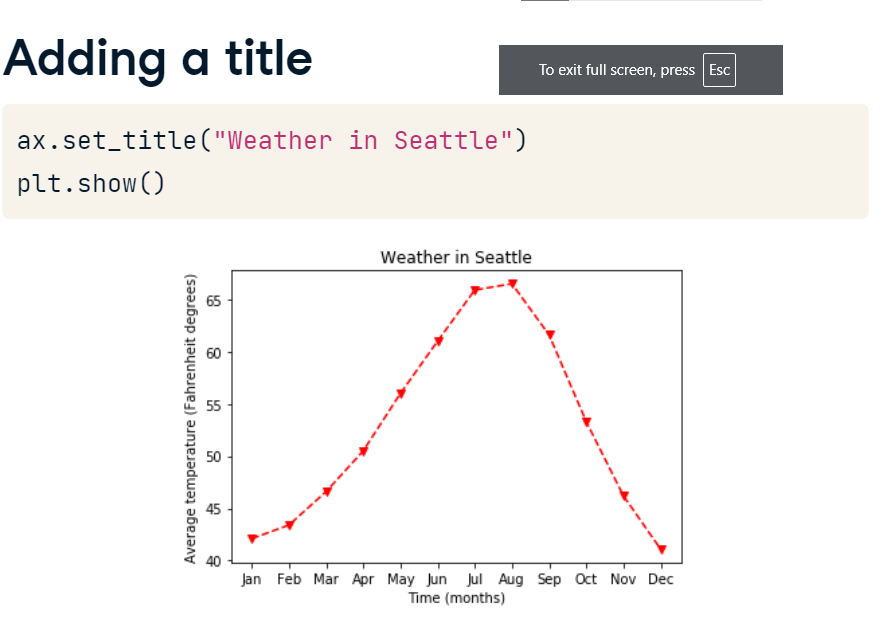
**Setting the y axis label**

Similarly, a set-underscore-ylabel method customizes the label that is associated with the y-axis. Here, we set the label to indicate that the height of the line in each month indicates the average temperature in that month.



**Adding a title**

Finally, you can also add a title to your Axes using the set-underscore-title method. This adds another source of information about the data to provide context for your visualization.

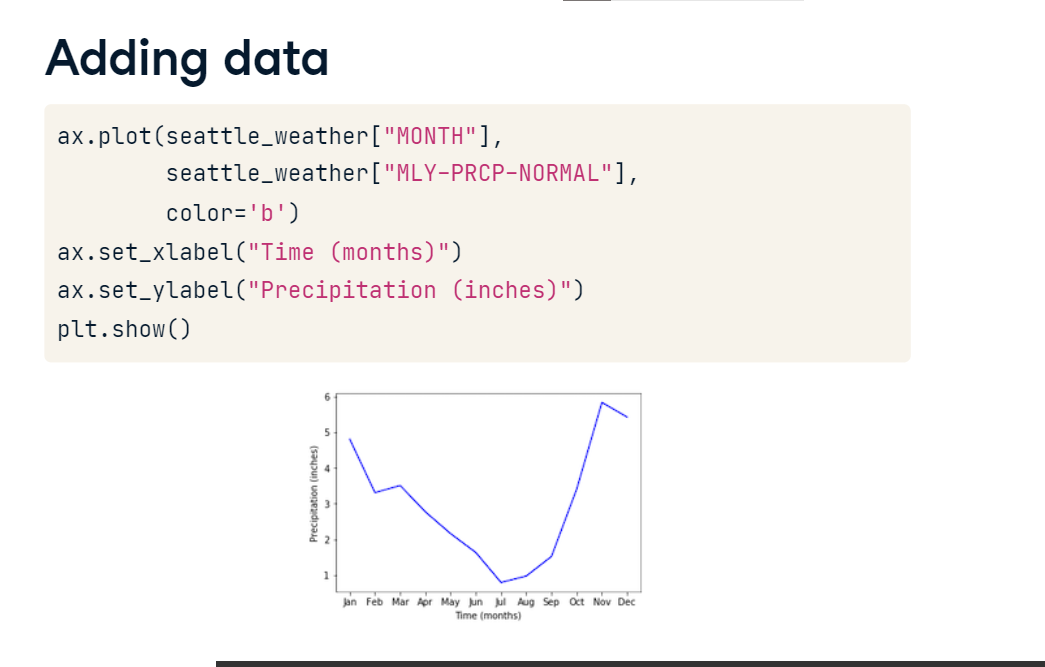


**Small multiples**

In some cases, adding more data to a plot can make the plot too busy, obscuring patterns rather than revealing them.

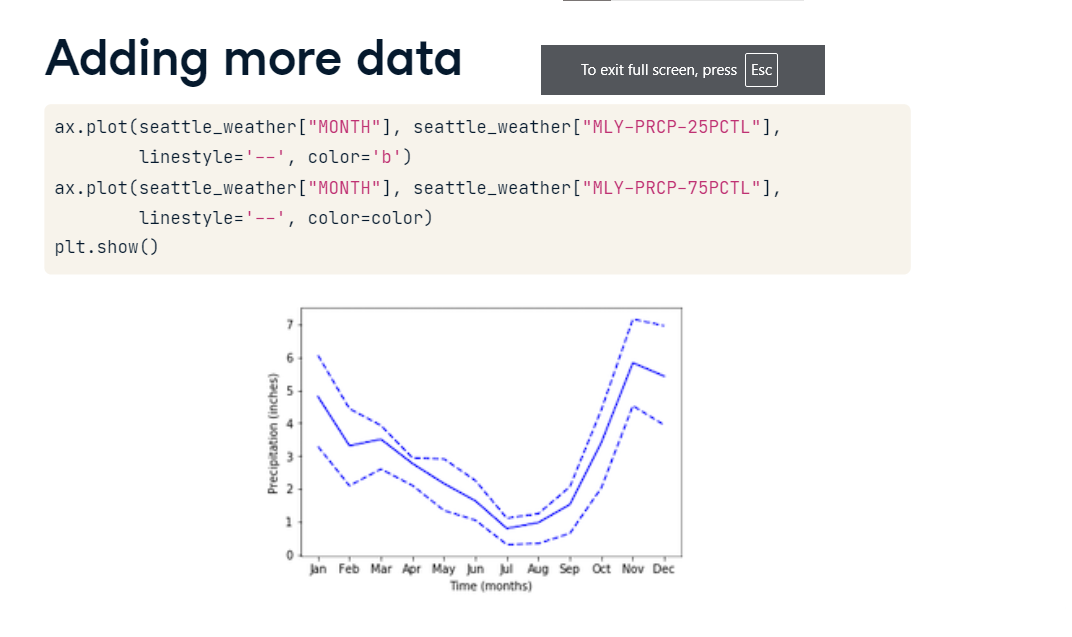
**Adding data**

For example, let's explore the data we have about weather in Seattle. Here we plot average precipitation in Seattle during the course of the year. But let's say that we are also interested in the range of values.



**Adding more data**

We add the 25th percentile and the 75th percentile of the precipitation in dashed lines above and below the average. What would happen if we compared this to Austin?



**And more data**

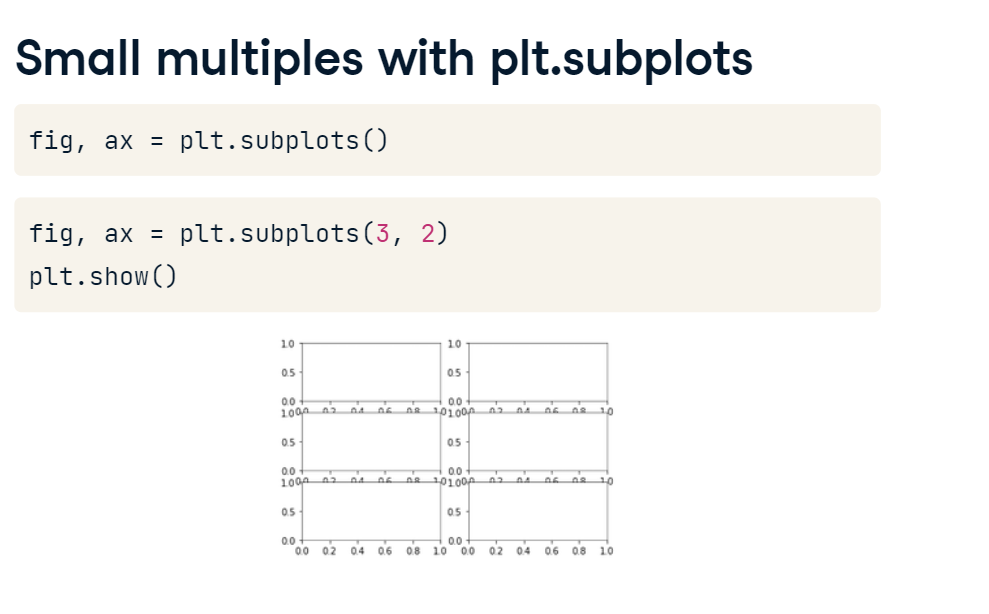
This code adds the data from Austin to the plot. When we display the plot, it's a bit of a mess. There's too much data in this plot. One way to overcome this kind of mess is to use what are called small multiples. These are multiple small plots that show similar data across different conditions. For example, precipitation data across different cities.



**Small multiples with plt.subplots**

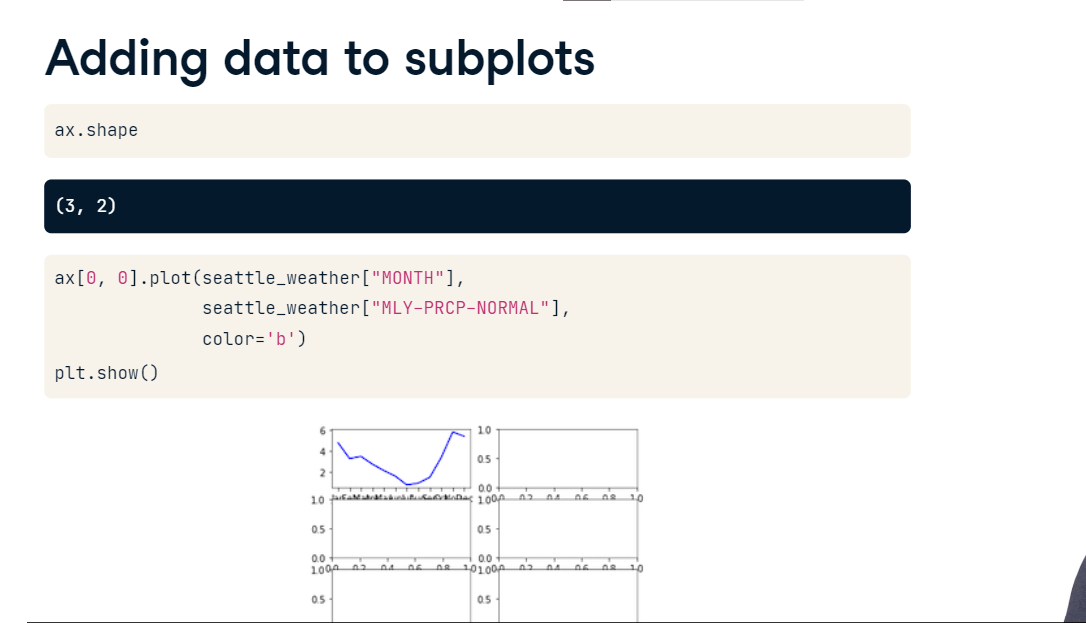
01:17 - 02:08

In Matplotlib, small multiples are called sub-plots. That is also the reason that the function that creates these is called subplots. Previously, we called this function with no inputs. This creates one subplot. Now, we'll give it some inputs. Small multiples are typically arranged on the page as a grid with rows and columns. Here, we are creating a Figure object with three rows of subplots, and two columns. This is what this would look like before we add any data to it. In this case, the variable ax is no longer only one Axes object.



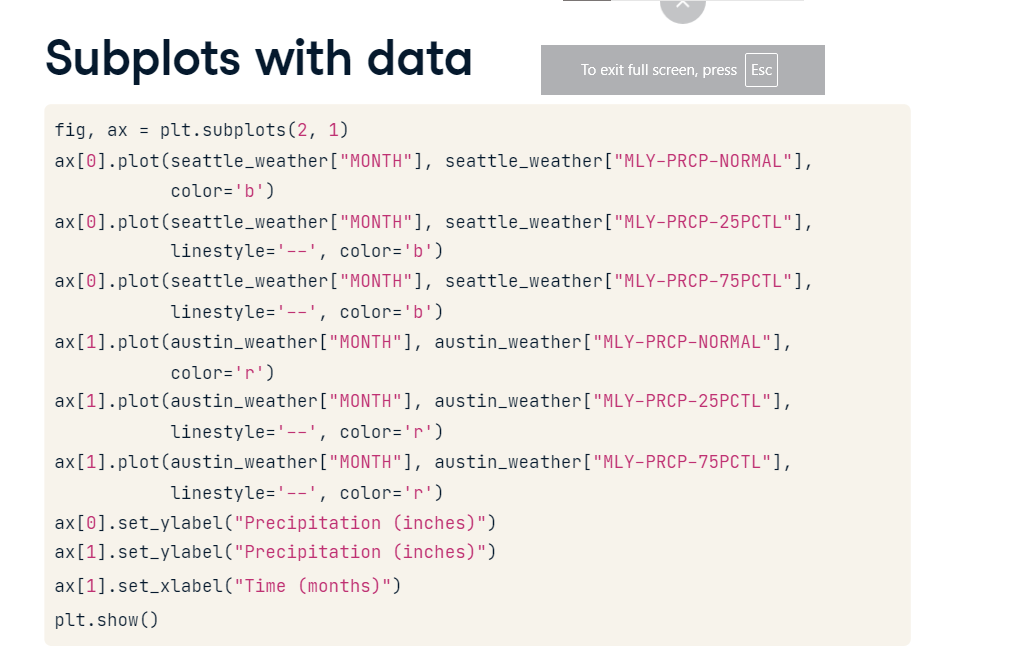
**Adding data to subplots**

Instead, it is an array of Axes objects with a shape of 3 by 2. To add data, we would now have to index into this object and call the plot method on an element of the array.



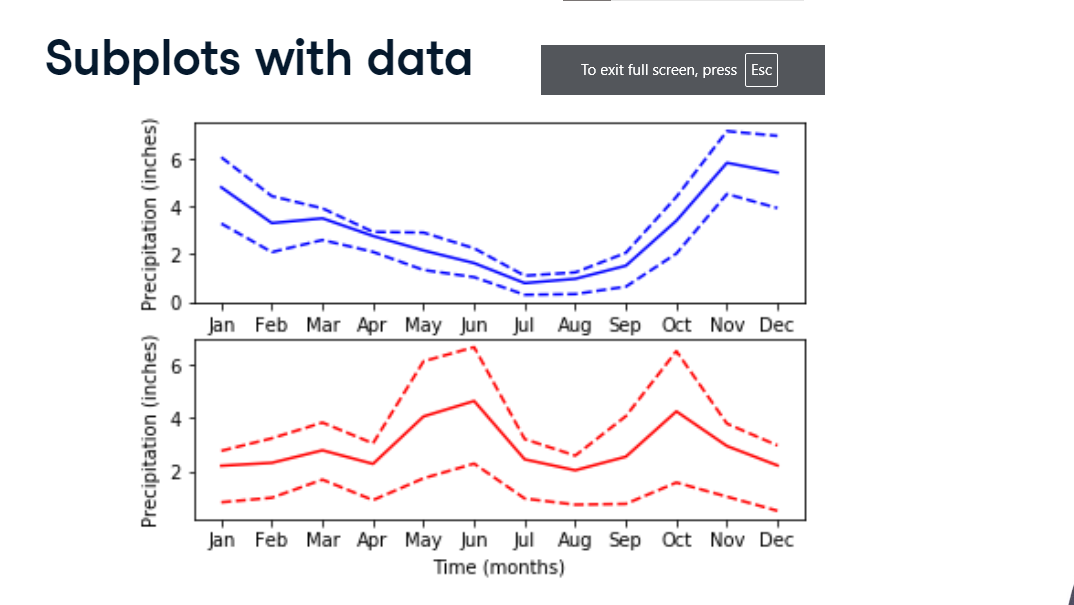
**8. Subplots with data**

There is a special case for situations where you have only one row or only one column of plots. In this case, the resulting array will be one-dimensional and you will only have to provide one index to access the elements of this array. For example, consider what we might do with the rainfall data that we were plotting before. We create a figure and an array of Axes objects with two rows and one column. We address the first element in this array, which is the top sub-plot, and add the data for Seattle to this plot. Then, we address the second element in the array, which is the bottom plot, and add the data from Austin to it. We can add a y-axis label to each one of these. Because they are one on top of the other, we only add an x-axis label to the bottom plot, by addressing only the second element in the array of Axes objects. When we show this,



**Subplots with data**

we see that the data are now cleanly presented in a way that facilitates the direct comparison between the two cities. One thing we still need to take care of is the range of the y-axis in the two plots, which is not exactly the same. This is because the highest and lowest values in the two datasets are not identical.



**Sharing the y-axis range**

To make sure that all the subplots have the same range of y-axis values, we initialize the figure and its subplots with the key-word argument sharey set to True. This means that both subplots will have the same range of y-axis values, based on the data from both datasets. Now the comparison across datasets is more straightforward.

